Design of a Transport Triggered Architecture Processor for Flexible Iterative Turbo Decoder

Shahriar Shahabuddin, Janne Janhunen, and Markku Juntti

Department of Communications Engineering and Centre for Wireless Communications, University of Oulu, Finland.

{sshahabu, janne.janhunen, markku.juntti}@ee.oulu.fi

Abstract—**In order to meet the requirement of high data rates for the next generation wireless systems, the efficient implementation of receiver algorithms is essential. On the other hand, the rapid development of technology motivates the investigation of programmable implementations. This paper summarizes the design of a programmable turbo decoder as an application-specific instruction-set processor (ASIP) using Transport Triggered Architecture (TTA). The processor architecture is designed in such manner that it can be programmed to support other receiver algorithms, for example, decoding based on the Viterbi algorithm. Different suboptimal maximum *a posteriori* (MAP) algorithms are used and compared to one another for the soft-input soft-output (SISO) component decoders in a single TTA processor. The max-log-MAP algorithm outperforms the other suboptimal algorithms in terms of latency. The design enables the designer to change the suboptimal algorithms according to the bit error rate (BER) performance requirement. Unlike many other programmable turbo decoder implementations, quadratic polynomial permutation (QPP) interleaver is used in this work for contention-free memory access and to make the processor 3GPP LTE compliant. Several optimization techniques to enable real time processing on programmable platforms are introduced. Using our method, with a single iteration 31.32 Mbps throughput is achieved for the max-log-MAP algorithm for a clock frequency of 200 MHz.**

I. INTRODUCTION

The turbo coding scheme [1] has been adopted for the air interface standard called Long Term Evolution (LTE), that has been defined by the 3rd Generation Partnership Project (3GPP) [2]. The decoding algorithm for the component decoder is not specified by 3GPP. Different suboptimal algorithms have been used for the turbo decoding and new solutions with less complexity are still being invented quite frequently. A programmable implementation simplifies the necessary support for different suboptimal algorithms of component decoders and different interleaving methods.

The turbo decoding algorithm is still one of the most computationally intensive parts of the wireless receiver. The software implementations provide the required flexibility to support multistandard solutions, but requires a careful design to achieve the target throughput. On the other hand, the hardware implementations provide high throughput, but the development time is not as rapid as processor based implementations. Programmable accelerators, which enable software-hardware co-design method might be an attractive solution to overcome these bottlenecks. The design of software and

hardware together to grind out the best performance and ensure programmability is not a straightforward task. The designer needs a very efficient tool, which can be used to design the processor easily for a particular application.

In this paper, we propose a design of a processor based on the Transport Trigger Architecture (TTA) for a flexible turbo decoder. TTA is a very good processor template for a programmable application-specific instruction-set processor (ASIP). The TTA based codesign environment (TCE) tool enables the designer to write the application with high level language and design the target processor in a graphical user interface at the same time [3].

The turbo decoder based on the transport triggred architecture was implemented by Salmela *et al.* [4]. The throughput of the processor was comparable to those of the turbo decoders based on a pure hardware design, but was less flexible in comparison with the design presented in this paper. The interleaver of the processor was not presented according to the 3GPP LTE standard.

The focus of the turbo decoder processor design in this paper is flexibility. The quadratic permutation polynomial (QPP) interleaving pattern have been used for the interleaver block. The comparison with the other suboptimal algorithms like log-MAP, constant-log-MAP and linear-log-MAP has also been presented.

The rest of the paper is organized in the following way: In Section II, an overview of the turbo decoder has been presented. In Section III, the simplification techniques of the turbo decoding algorithm and its implementation in high level language is presented. The processor design has been presented in Section IV. In Section V, we will present the throughput results of the design. The conclusion is given in Section VI.

II. TURBO DECODER

A. Turbo Decoder Structure

The turbo decoder consists of two soft-input soft-output (SISO) decoders, with interleavers and de-interleavers between them as shown in Fig. 1. The inputs of the turbo decoder come from the soft demodulator, which produces the log-likelihood ratios (LLR) for the systematic bits and parity bits. The LLRs of the systematic bit, LuI and first parity bits, LcI1 goes to the first SISO decoder. The SISO decoder produces soft outputs

based on these LLRs. These soft outputs are used in the second SISO decoder as the additional information. The inputs of the second SISO decoder are the LLRs coming from the systematic bits, second parity bits denoted by LcI2 and output of the first SISO decoder. The LLRs of the systematic bits are scrambled this time with the same interleaving pattern used at the encoder. Similarly, the soft outputs coming from the first SISO decoder are scrambled also with the same interleaving pattern, which are used as *a priori* values for the second SISO decoder.
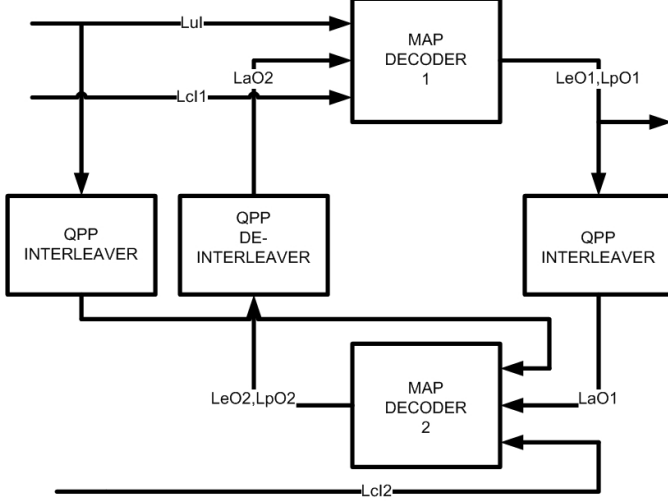

Fig. 1. Block diagram of the turbo decoder.

The heart of the turbo coding is the iterative decoding procedure. The output of the second SISO decoder does not produce the hard outputs immediately, but the soft output is used again in the first SISO decoder for more accurate approximation. The process continues in a similar fashion in an iterative manner. A single iteration by both the first and the second SISO decoder is referred to as a full iteration. On the other hand, the operation performed by a single SISO decoder can be referred to as a half iteration. At the beginning of the first iteration, the *a priori* values are set at zero. Six to eight full iterations are used to achieve sufficient performance [1].

B. MAP algorithm for Component Decoder

The MAP algorithm for the component decoder applied here has been proposed by Benedetto *et al.* [5]. The algorithm can be stated like:

1. Initialize the values of the forward state metric as $\alpha_0(s) = 0$ if $s = S_0$ and $\alpha_0(s) = -\infty$ otherwise.

2. Calculate all the forward state metric of the same window through the forward recursion according to

$$
\begin{aligned}
\alpha_k(s) = \max{}^*(\alpha_{k-1}[s^S(e)] + u(e)LuI[k-1] \\
+ c_1(e)LcI1[k-1] + c_2(e)LcI2[k-1]).
\end{aligned} \tag{1}
$$

3. Initialize the values of the backward state metric as $\beta_n(s) = 0$ if $s = S_n$ and $\beta_n(s) = -\infty$ otherwise.

4. Calculate all the backward state metric of the same window through the backward recursion as

$$
\begin{aligned}
\beta_k(s) = \max{}^*(\beta_{k+1}[s^E(e)] + u(e)LuI[k+1] \\
+ c_1(e)LcI1[k+1] + c_2(e)LcI2[k+1]).
\end{aligned} \tag{2}
$$

5. The LLR values for the information and both parity bits can be calculated as following:

$$
\begin{aligned}
LLR(.;O) = \max{}^*(\alpha_{k-1}[s^S(e)] + c_1(e)LcI1[k-1] \\
+ c_2(e)LcI2[k+1] + \beta_{k+1}[s^E(e)]).
\end{aligned} \tag{3}
$$

The forward metric and backward metrics increase in each step and that is why the forward and backward metrics need to be normalized to avoid memory overflow.

The decoding is done in smaller windows so that the decoding process can be done in parallel and the decoder does not have to wait for the whole block to arrive before starting the decoding process. This windowing is sometimes referred to as a sliding window method.

Based on the definition of the max* function, the suboptimal algorithms can be changed for the MAP algorithm. As for an example, for constant-log-MAP algorithm it can be expressed as

$$
\max{}^*(x,y) = \max(x,y) + \begin{cases} 0 & if|y-x| > T \\ C & if|y-x| \le T. \end{cases} \tag{4}
$$

We considered four suboptimal MAP algorithms, which are Max-log-MAP, constant log MAP, linear-log-MAP and an approximation of log-MAP algorithm. A more detailed overview of these algorithms can be found in [6], [7] and [8].

C. QPP Interleaver

The QPP interleaver has been adopted for the 3GPP LTE standard [3]. Unlike the earlier 3G interleavers, the QPP interleaver is based on algebraic constructions. The QPP interleaver can be expressed by a simple mathematical formula. The relationship between the output index x and the input index $f(x)$ can be derived from this formula given below

$$
f(x) = (f_2x^2 + f_1x)\mathrm{mod}(N). \tag{5}
$$

The values of the parameters f_1 and f_2 of (5) are integers and depend on the block size N. For a different N, a different set of parameters have been defined in [9]. The block sizes are divisible by 4 and 8. The value of f_1 is an odd number while the value of f_2 is an even number. Several algebric properties of QPP interleaver have been listed in [10].

III. DESIGN IN HIGH LEVEL LANGUAGE

The MAP algorithm described above is complex and needs to be simplified. The repeated calculations should be avoided, which in turn reduces the latency of the processor. The reduced calculations used in this work will be described in the following section. The reduced design has been tested in a link level simulator using Matlab. The Matlab design is converted to C language in such a way so that it can be efficiently implemented in the TCE tool.
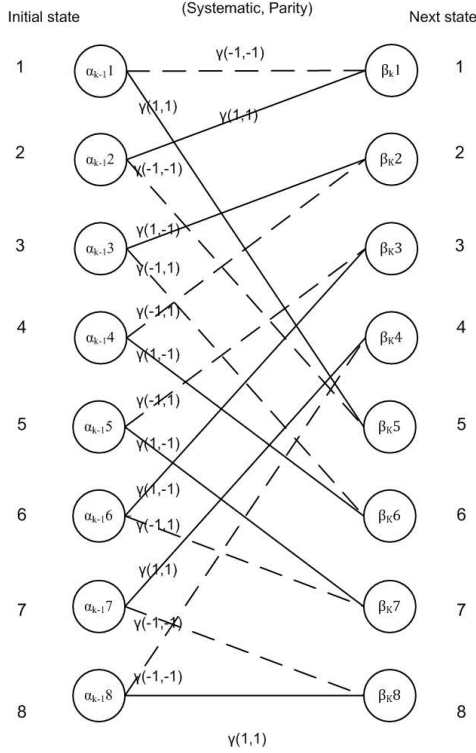
Fig. 2. Trellis of 3GPP turbo code.

metrics have been calculated first in this design by following the simplified equation. The normalization technique is done following the technique suggested by Valenti *et al.* [12]. Typically, the normalization is done in every step by subtracting the minimum values of forward or backward metrics of the same column from every forward or backward metric values. They suggested subtracting all the values from the first values of the columns. Thus, there is no need to store the first row and this constitutes 12.5 percent savings in memory compared with the other normalization methods.

Only the forward metrics are saved for the calculation of the LLRs. The backward metrics and LLR calculations are done together. As soon as the backward metric calculations of a single column are finished, the LLRs are calculated in parallel with the help of the stored forward metric at the same time. So there is no need to save the backward matrics [12].

Originally, α and β are two-dimensional matrices of size 6144×8 as the input block of 6144 information bits have been taken into consideration. The processor needs to do more calculation for accessing these two dimensional matrices. Instead of using a single 6144×8 matrix, seven vectors of 6144 elements have been used. As mentioned above, the eighth vector is not needed due to the normalization strategy adopted in the used method.

The turbo decoder description in C needs to be efficient to fulfill the latency requirement. A lot of data dependencies can be seen from the turbo decoder algorithm. So, writing efficient code is needed for the utilization of computing resources available.

Assuming thatenough register files are available, efficient code is written to reduce memory accesses by holding the values inside the register files. It is also possible to eliminate the need for accessing the forward metric vectors constantly. Instead, the temporary variables are used to hold the values inside the registers. The code for the backward metric and the LLR calculations have been written in the same manner. The calculations are done is several smaller loops to follow the sliding window technique and to enable parallel calculations.

A look-up table, holding the values of the f_1 and f_2 parameters, is used to implement the QPP interleaver. The rest of the calculations are quite straightforward.

There are 16 branch metric computations between two states for forward metric, backward metric and LLR calculations. From the trellis structure of the 3GPP turbo code it can be seen that four calculations of branch metric are being repeated to result in total sixteen calculations. Instead of multiplying with 1 and -1, the branch metrics can be calculated directly based on these four calculations,

$$\gamma_1 = LuI + LcI1 + LcI2$$
$$\gamma_2 = -LuI - LcI1 + LcI2$$
$$\gamma_3 = LuI + LcI1 - LcI2$$
$$\gamma_4 = -LuI - LcI1 - LcI2,$$

(6)

where γ_4 can be represented as $-\gamma_1$ and γ_3 can be represented as $-\gamma_2$. For every forward metric, the backward metric and the LLR, calculations of two branch is sufficient. As an example, the backward metric calculations are shown as,

$$\beta_1(k) = \max{}^*(\beta_1(k+1) - \gamma_1, \beta_5(k+1) + \gamma_1)$$
$$\beta_2(k) = \max{}^*(\beta_1(k+1) + \gamma_1, \beta_5(k+1) - \gamma_1)$$
$$\beta_3(k) = \max{}^*(\beta_6(k+1) + \gamma_2, \beta_2(k+1) - \gamma_2)$$
$$\beta_4(k) = \max{}^*(\beta_6(k+1) - \gamma_1, \beta_2(k+1) + \gamma_1)$$
$$\beta_5(k) = \max{}^*(\beta_1(k+1) - \gamma_1, \beta_5(k+1) + \gamma_1)$$
$$\beta_1(k) = \max{}^*(\beta_1(k+1) - \gamma_1, \beta_5(k+1) + \gamma_1)$$
$$\beta_1(k) = \max{}^*(\beta_1(k+1) - \gamma_1, \beta_5(k+1) + \gamma_1)$$
$$\beta_1(k) = \max{}^*(\beta_1(k+1) - \gamma_1, \beta_5(k+1) + \gamma_1).$$

(7)

A similar idea has been presented in [11] where a simplified version of max-log-MAP has been proposed. The forward

IV. TRANSPORT TRIGGERED ARCHITECTURE PROCESSOR

A. General Functional Units

A part of the processor designed for the turbo decoder is illustrated in Fig. 3. For readability, the whole processor figure is not given. The blocks on the upper part of the figure represent the functional units and register files of the processor. The black horizontal straight lines represent the buses of the processor. The vertical rectangular blocks represent the sockets. The connection between functional units and buses is illustrated by black spots in the sockets.

The fixed point processor includes load/store unit (LSU), arithmetic logic unit (ALU), global control unit (GCU) and register files. Based on the resource requirements in high level
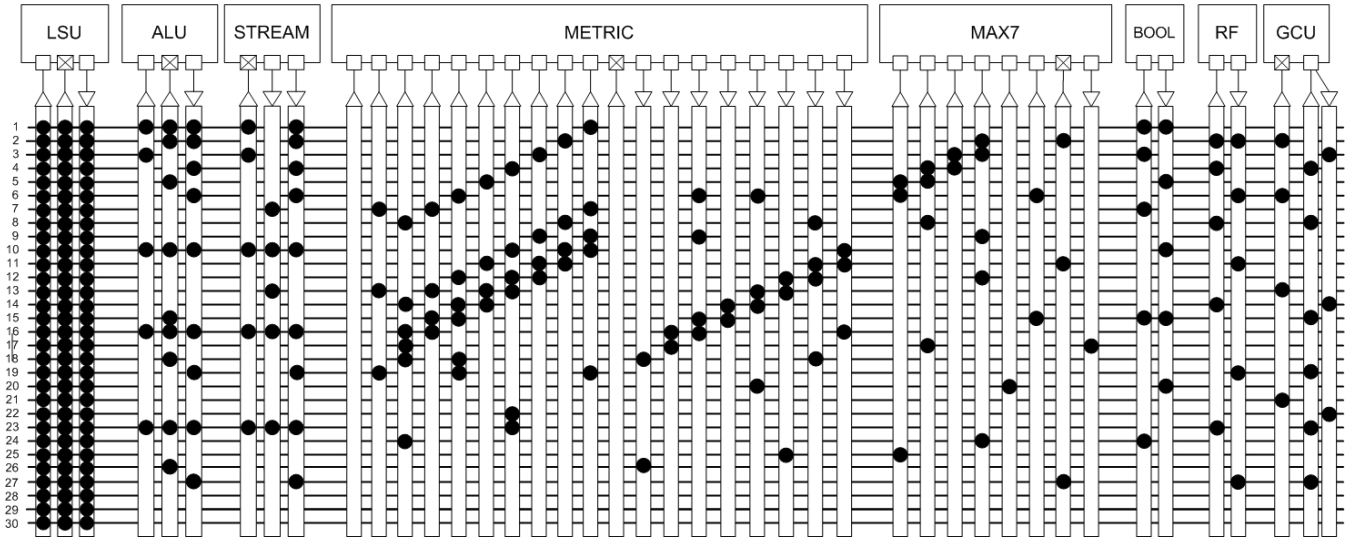
Fig. 3. Implemented processor with reduced number of functional units.

language, more functional units and register files have been added.

The LLR inputs have been read from a first-in-first-out (FIFO) memory buffer by using the functional unit called STREAM. The STREAM units can read every input sample in one clock cycle. Eight STREAM units have been used to get the input LLRs simultaneously. The STREAM units are used to implement the sliding window technique that helps to decode the input block in smaller parts parallely. One STREAM unit has been used to write the output LLRs in the memory buffer.

Three LSU units have been used to support the memory accesses. The LSU units are used to read and write memory. The memory can be read in three clock cycles and can be written in a single cycle.

The ALU unit has been used to perform the basic arithmetic operations like addition, subtraction etc. Operations like shifting right or left are also included in ALU.

B. METRIC Special Function Unit

Like the Viterbi decoder, the turbo decoder needs numerous add-compare-select operations. One way to design the processor for these algorithms is to use the appropriate number of adder and maximum selection units. Another way is to design a special function unit to calculate all the necessary next state metrics based on the earlier state metrics and branch metrics.

The latter way is followed in our design and a special function unit named METRIC has been designed with twelve inputs and eight outputs. Eight of these inputs correspond to the forward metrics in case of forward metric calculations and backward metrics in case of backward metric calculations. Three of these inputs correspond to the *a priori* LLR, the LLR of systematic bits and the LLR of parity bits, respectively. One input is used to select different suboptimal algorithms and this input is named *mode* in this paper.

The forward and backward metrics calculations are not the same. The inputs and outputs of the unit need to be selected carefully because the same unit has been used for both forward and backward metric computations. Using the same unit for forward and backward metric computations is possible because of the trellis structure of a 3GPP turbo code.

The design of the Metric unit for one butterfly pair in the trellis is shown in Fig. 4. The values of branch metrics to calculate $\alpha_2(k)$ and $\beta_3(k)$ are the same. Likewise, the values of branch metrics to calculate $\alpha_6(k)$ and $\beta_4(k)$ are also the same. It is possible to calculate these forward and backward metrics by changing the input values of the earlier state metrics. This is true for all the rest of the butterfly pairs in the trellis. The METRIC unit uses this technique of changing the orders of the input metrics for the backward metrics calculation, but using the same function unit.

The METRIC unit supports four operations. These four operations are branch metric calculations for max-log-MAP, linear-log-MAP, constant-log-MAP and log-MAP respectively. The latency of these operations are different because these operations use different codes for different algorithms.

C. MAX7 Special Function Unit

Another special function unit named MAX7 has been designed, which is used to calculate the output LLRs after the forward and backward metrics computation. This unit also has four options for four suboptimal algorithms. For the max-log-MAP *mode*, the MAX7 unit takes seven inputs and finds the maximum of these inputs. The LLR calculation uses the maximum value of seven computations in this case. The addition of the forward metric, branch metric and next state backward metric represent these computations. It is possible to reduce one computation because of the normalization method chosen in this design. We have given the added values of these seven computations as inputs to this unit and used the output as
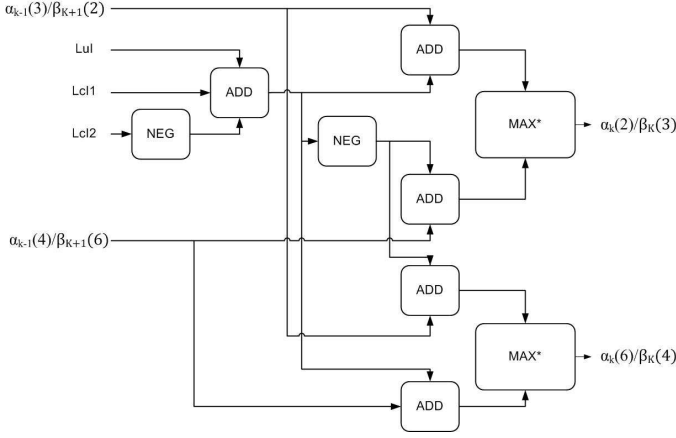
Fig. 4.   METRIC unit for a single butterfly pair.

the LLR. The latency of this unit is different for four different algorithms.

D. Registers

Several register files have been used to save the intermediate results. In terms of the power consumption, registers can be more expensive than memory, but to meet the latency requirement register files are needed. A single Boolean register file has been included in the processor design.

V. RESULTS AND DISCUSSION

The designed processor takes 39,226 clock cycles to process three blocks of 6,144 samples for a full iteration for the max-log-MAP mode. According to the 3GPP interleaver specifications, the size of the block could be up to of 6,144 samples and that is the reason the size of the input block is chosen as 6,144.

When we assume a clock of 200 MHz, then the throughput of one iteration for the max-log-MAP could be calculated as,

$$\text{Throughput} = \frac{6,144 \text{ bits} \times 200 \text{ MHz}}{39,226 \text{ clock cycles}}$$
$$= 31.32 \text{ Mbps.}$$

The clock cycle needed for a single trellis stage of the max-log-MAP can be calculated as,

$$\text{Cycle}_{Stage} = \frac{39,226}{2 \times 6,144}$$
$$= 3.19 \text{ cycles.}$$

TABLE I
NUMBER OF CLOCK CYCLES FOR A SINGLE ITERATION

Modes	Algorithm	Clock Cycle
1	max-log-MAP	39,226
2	linear-log-MAP	103,436
3	constant-log-MAP	184,454
4	log-MAP	834,253

It can be seen from Table I that the processor takes more clock cycles if max-log-MAP algorithm is not used. The

reason lies in the fact that the constant-log-MAP, linear-log-MAP and log-MAP algorithms need to invoke some conditional statements resulting in branches in execution. Thus, the latency increases compared to the max-log-MAP algorithm. A less complicated approximation of log-MAP has been used in this design. However, the correction term of the log-MAP increases the latency significantly compared to the other three suboptimal algorithms even after the simplification. Linear-log-MAP is slower than the constant-log-MAP because of some multiplication operations are needed. The log-MAP algorithm provides better BER performance than the other algorithms. This log-MAP can be used in cases when the latency requirements are not strict and a high BER performance is needed.

The buses of the processor are perfectly utilized to achieve the best possible result due to the perfect scheduling. The number of some of the operations during the algorithm execution has been summarized in Table II.

TABLE II
NUMBER OF OPERATIONS

Operation	# of OPS
ADD	339,009
SUB	98,304
MUL	45,234
MAX7	24,567
BRANCH	24,576
STREAM	24,556

The number of addition operations does not only represent the addition for the algorithm, but for several other purposes like loop indexing for the code. The subtraction operations are due to the normalization and the multiplication is used for the correction term calculation of linear-log-MAP and for the QPP interleaving sequence generation.

The throughput can be increased using dedicated special function units working as accelerators for the processor. On the other hand, the more special function units are used, the less flexible the processor becomes. As an example, a special function unit dedicated to calculate the backward metric and the LLRs could be designed to reduce latency. However, it might be useless for some other functionalities. A common special function unit to calculate both forward and backward metric could be better utilized in this case.

The TTA processor designed here will work for the Viterbi decoding with reasonable throughput. The BRANCH unit for the max-log-MAP mode is able to calculate the add-select-compare operations needed for the Viterbi decoding.

A comparison with different other programmable implementations of turbo decoder has been presented in Table III. Our proposed processor provides very good throughput compared to most of the programmable implementations. The processor throughput is lower than the TTA processor of [4] for the same clock frequency of 200 MHz, because of the complex structure of QPP interleaver and de-interleaver.

TABLE III
PROGRAMMABLE PROCESSORS

Reference	Architecture	Algorithm	Throughput
[13]	Motorola	max-log-MAP	243 Kbps
[12]	Intel Pentium	max-log-MAP	366 Kbps
[14]	TMS320C6201 DSP	max-log-MAP	2 Mbps
[15]	VLIW ASIP	max-log-MAP	5 Mbps
[4]	TTA proc. for UMTS	max-log-MAP	14.1 Mbps
Proposed	TTA proc.	max-log-MAP	31.21 Mbps
[4]	TTA proc.	max-log-MAP	98 Mbps

VI. CONCLUSION

The paper discussed the design issues of a turbo decoder on a TTA processor. The reference turbo decoder design has been simulated on a Matlab link level simulator. The design is then converted and optimized in C language and mapped on the TTA processor using TCE tool. The design shows the promise of the possibility of designing several decoding techniques on a single TTA processor. As the turbo decoding algorithm is very complex, it is very difficult to achieve the LTE target throughput even with pure hardware designs. The parallel multi-core turbo decoder is the natural choice for the next generation wireless systems. The target throughput could also be reached by multi-core TTA processor. The flexibility gained from that processor could provide very interesting results and would be a fruitful direction for future research.

VII. ACKNOWLEDGEMENT

REFERENCES

[1] C. Berrou, A. Glavieux, P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: turbo-codes," in *IEEE International Conference on Communication*, vol. 2, pp. 1064-1070, Geneva, Switzerland, May 1993.

[2] Multiplexing and channel coding, 3GPP TS 36.212 version 10.5.0.

[3] P. Jääskeläinen, V. Guzma, A. Cilio, T. Pitkänen, and J. Takala, "Codesign toolset for application-specific instruction-set processors," in *Multimedia on Mobile Devices 2007*, vol. 6507 of *Proceedings of SPIE* pp. 1-11, San Jose, Calif, USA, January 2007.

[4] P. Salmela, H. Sorokin, and J. Takala, "A programmable max-log-MAP turbo decoder implementation," *Hindawi VLSI Design*, vol. 2008, pp 636-640, 2008.

[5] S. Benedetto, G. Montorsi, D. Divsalar, and F. Pollara, "A soft-input soft-output maximum a posteriori (MAP) module to decoder parallel and serial concatenated codes," in *JPL TDA Progr. Rep.*, vol. 42-127, pp. 1-20, Jet Propulsion Lab., Pasadena, CA, 1996.

[6] P. Robertson, P. Hoeher, and E. Villebrun, "Optimal and suboptimal maximum a posteriori algorithms suitable for turbo decoding," in *European Trans. on Telecommun.*, vol. 8, pp. 119-125, Mar./Apr. 1997.

[7] B. Classon, K. Blankenship, and V. Desai, "Turbo decoding with the constant-log-MAP algorithm," in *Proc., Second int. Symp. Turbo Codes and Related Appl.*, pp. 467-470, Brest, France, Sept. 2000.

[8] J.-F. Cheng, and T. Ottosson, "Linearly approximated log-MAP algorithms for turbo coding," in *Proc., IEEE Veh. Tech. Conf.*, Houtson, TX, May 2000.

[9] A. Nimbalker, Y. W. Blankenship, B. K. Classon, and K. T. Blankenship, "ARP and QPP Interleavers for LTE Turbo coding," in *IEEE Wireless Communications and Networking Conference*, pp. 10321037, April 2008.

[10] Y. Sun and J. R. Cavallaro, "Efficient hardware implementation of a highly-parallel 3GPP LTE, LTE-advance turbo decoder," in *Integr.,VLSI J.*, vol. 44, no. 1, pp. 1-11, 2010.

[11] P. Salmela, T. Järvinen and J. Takala, "Simplified max-log-MAP decoder structure," in *Proceedings of the 1st Joint Workshop on Mobile Future and the Symposium on Trends in Communications (SympoTIC '06)*, pp. 1-11, San Jose, Calif, USA, January 2007.

[12] M. C. Valenti, J. Sun, "The UMTS turbo code and an efficient decoder implementation suitable for software defined radios," in *International Journal on Wireless Information Networks*, vol. 8, no. 4, pp. 203-216, Oct. 2001.

[13] H. Michel, A. Worm, M. Munch, and N. Wehn, "Hardware software trade-offs for advanced 3G channel coding," in *Proceedings of Design, Automation and Test in Europe*, 2002.

[14] Y. Song, G. Liu and Huiyang, "The implementation of turbo decoder on DSP in W-CDMA system," in *International Conference on Wireless Communications, Networking and Mobile Computing*, pp. 1281-1283, Dec. 2005.

[15] P. Ituero and M. López-Vallejo, "New schemes in clustered VLIW processors applied to turbo decoding," in *Proceedings of International Conference on Application-Specific Systems, Architectures and Processors (ASAP '06)*, pp. 291-296, Steamboat Springs, Colo, USA, September 2006.